An Ontology-Based Autonomic System for Improving Data Warehouses by Cache Allocation Management

Vlad Nicolicin-Georgescu, Vincent Benatier, Rémi Lehn, Henri Briand

An Ontology-Based Autonomic System for Improving Data Warehouses by Cache Allocation Management

Vlad Nicolicin Georgescu and Vincent Benatier
SP2 Solutions, www.sp2.fr, vladgeorgescun@sp2.fr
Remi Lehn and Henri Briand
LINA CNRS 6241 - COD Team – Ecole Polytechnique de l'Université de Nantes

Abstract

With the increase in the amount and complexity of information, data warehouse performance has become a constant issue, especially for decision support systems. As a consequence, decision experts are faced with the management of all this information, and thus realize that special techniques are required to keep good performances. This paper proposes an approach to data warehouse systems improvement based on Autonomic Computing. The idea is that by rendering certain tasks autonomic, such as the configuration of cache memory allocations between groups of data warehouses, we ensure a better data warehouse management at a lower cost and save substantial decision experts' time that can be used on higher level decision tasks.

1 Introduction

Decision Support Systems are defined as computerized systems whose main goal is to analyze a series of facts and give various propositions for actions regarding the facts involved [Druzdel and Flynn (1999)]. The process of decision making in enterprises based on such systems is also known as Business Intelligence. This concept is very well applied by large enterprises. Via this process, they specifically focus on their data warehouse efforts. The problem is that data warehouses usually become fast very large and complex, thus their performances become rapidly an issue. This is why between 70 and 90% [Frolick and Lindsey (2003)] of the enterprises consider that their data warehouse efforts is inefficient, as in many cases, the large amount of data involved becomes unusable. In many of these cases, the cause is bad management or costs that are too high to sustain.

One of the main problems that lead to this is common resource sharing between data warehouses. The resources are usually limited either by financial costs or by architectural considerations. Consider the following real example, to emphasize the problematic. An enterprise has a special server for its data warehouses. In total, a group of 50 data warehouses that share the same RAM memory is deployed on this server. Each of the data warehouse requires at least 20 GB of RAM to have good performances (i.e. the query average time is under a second). So there is a need for at least 1TB of RAM (ignoring all other RAM requirements of the server). First, the costs of having 1TB of RAM on server are financially high (~ 40000 EUR[1]). Second, if the enterprise is ready to cover these costs, suppose the server has an architecture that enables a maximum of 16GB to be installed. Also the migration of some data warehouses on another server would be too expensive and too complicated. An option is to compromise, asking each time an expert to re-configure the memory allocation for each of the data warehouse. In a short time after this is done, with the evolution of the data warehouses' size or if new data warehouses are added or some become obsolete, the problem reappears and the same action must be taken, over and over again.

Based on the example above we can intuitively see a simple solution: enable autonomic tasks that reconfigure the memory allocations, instead of asking a human expert each time to intervene (human resources are the most expensive, and not always provide the optimal results). This is easy to be said but it is hard to formalize, due to two main issues.

First, how to formally represent the group of data warehouses along with the knowledge involved in the decisions and actions of the expert? To do this, we differentiated three main types of information that needs to be formalized: a) architectural information (how a group of data warehouse is organized, the number of groups, how are they linked, etc.); b) configuration and performance information (how much memory each data warehouse needs, what performance is achieved with this allocation, etc.); and c) experience information that represents best practices and advices for the memory allocations (coming from editor documents, human experience, etc.). We present in this paper a formalization of the three types into a unified knowledge base, using ontologies [Gruber (1992)] and ontology based rules [Stojanovic et al.(2004)].

Second, having the information formalized, we need an organized form of rendering the autonomic process. To this end, IBM proposes a solution called Autonomic Computing [IBM (2001)]. It consists in the division of the actions that are taken when trying to provide autonomy to a process, corresponding to objective-specific phases and states. Autonomic concepts can be integrated in hierarchical organized systems, so each higher level aggregates what has been done to its sub levels. There are numerous autonomic computing based works that relate especially to problem resolution [Manoel et al.(2005)] or system administration [Barret et al.(2004)]. On the other hand, little has been done on data warehouse improvement.

[1] http://www.materiel.net/ctl/Serveurs/

So, we propose to use autonomic computing on the unified formalized knowledge base. Specifically, we treat a common configuration problem: cache memory allocation for a group of data warehouses (that share the same amount of common available RAM memory). The objective is to reach a better performance (in terms of query response times when extracting data from the data warehouse) with lower costs. The implication is that by increasing the amount of cached data, there are better chances that a request hits the cache; the response times in order to extract the data decrease, which translates in better performances. But, the whole amount of data obviously can't be put in the cache, and then we need a way to automatically determine and adjust the cache parameters.

Section 2 presents a view of data warehouse management through caches in the context of decision support systems. It presents the information that needs to be manipulated and how the division of this information can lead to a unified knowledge base representation. Section 3 presents how autonomic computing is used with managing data warehouse through caches. It presents how the knowledge base is integrated to permit autonomic tasks. It equally proposes two heuristics for cache allocation, based on the problematic described. Section 4 shows how we integrate the elements together using ontologies for the knowledge base representation and ontology based rules for the autonomic process. We provide some results obtained with our approach. In the conclusion we sum the work presented giving future directions and hoping that our work could help enterprises with their data warehouse efforts.

2 Data Warehouse and Cache Allocations

First of all, when speaking of data warehouse we usually make reference to a definition as a repository of an organization's electronically stored data and is designed to facilitate reporting and analysis [Inmon (2005)]. Managing a data warehouse includes the process of analyzing, extracting, transforming and loading data and metadata. As decisional experts, we know that once data warehouses are put in place, enterprises then base their decisions on the data that is stored within them. So a good organization in start and a good performance in time are the requirements of data warehousing.

We do not put into question the initial organization. We observed that in time data warehouse performances are constantly degrading up to a point where the system is no longer usable. One aspect of data warehouse performance is strongly related to the operation of data extraction which in turn depends on the query response times on the data. Obviously, the larger a data warehouse is, the more information it contains so we expect to have higher response times. Considering that some information is often more demanded than other, data warehouse management systems offer the possibility of keeping frequently accessed data in cache memories with the hypothesis that fetching data from the cache is greatly faster than fetching them from the persistent storage media. The problem occurs when confronted with groups of data warehouse on the same machine that share the same amount of memory. In decision support systems, such groups contain data of up to several thousand gigabytes. They cannot be all put into the cache, so solutions are required.

Although the problematic of performance improvement in data warehouses throughout caches is debated [Malik et al.(2008)], [Saharia and Babad (2000)] the issue is always addressed either through the physical design or the design of algorithms to determine which information is likely to be stored in cache memories. These solutions apply well when we focus on a single data warehouse.

So, what actually happens in enterprises is that the initial cache allocations remain the same throughout time. Whereas the quantity of data in the data warehouse increases, some of them are no longer used; there are new data warehouses that are constructed etc. Therefore there is a need for a dynamic system.

The first aspect of the system we propose and that we approach is knowledge formalization. In the example presented in the introduction, the expert in order to reallocate the memory makes use of several types of information. We propose to divide this information into three main types, detailing and exemplifying based on the Hyperion Essbase[2] business intelligence solution.

Architectural information corresponds to the organization of the groups of data warehouse. Figure 1 shows an example of a possible organization.

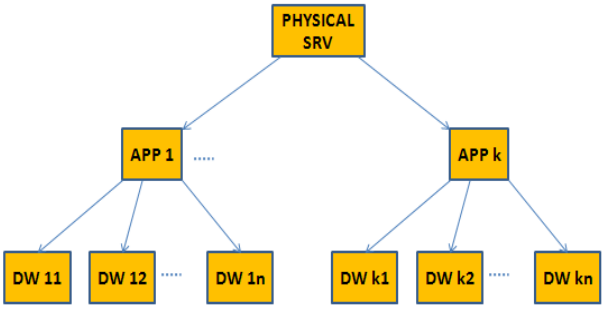


Figure 1 - Architectural organization for groups of data warehouse

Based on a decision support system simple organization, we can distinguish on top of the tree a Physical Server as the actual machine. Underneath, there are a number of Applications installed that share the RAM memory available on the server. And, in turn each application contains one or more data warehouses (Essbase cubes or bases), sharing the same memory. Each application is seen as a group of data warehouses, and then memory reallocation is done within each application.

Configuration and performance information contains all the indicators that reflect the actual characteristics and configuration of the data warehouses and the performances obtained with this configuration. For the characteristics and configuration we refer to the Essbase cubes. There are many characteristics, but for our example we take into consideration the following indicators:

- The size of each data warehouse represented by: the Index File size and the Data File size. This corresponds to the actual size that each data warehouse is occupying on a hard disk. A value of tens of GB for the two together is a frequently met characteristic.
- The values of three types of caches: Index, DataFile and Data Cache. Corresponding to the sizes presented before, they represent a percentage of the actual data files that can be kept in cache. Ideally, we should have the total of index file size in the index

[2]http://download.oracle.com/docs/cd/E10530_01/doc/epm.931/html_esb_dbag/frameset.htm?dstcache.htm

cache, and the total of data file size in the data and data file cache.

For the performance aspect, there are many indicators to take into consideration such as: query response times, calculation times, aggregation operation times, etc. We chose the query response time as a performance indicator as this is a frequently used measure [Saharia and Babad (2000)] of the system performance, and, it directly reflects the quality of the user experience in the decision system. It represents the time needed to extract data through a query from the data warehouse.

Experience and best practices information represent a more delicate subject in comparison with the first two information types. The main reason is that it comes from several different sources. Therefore the challenge is how to combine these sources into a single unified knowledge base. For instance, how to combine practices taken from an Essbase support document with practices that are part of the human experience and that are only known by the expert. We present here the formalization aspect, that is revised and validated by a human expert. In order to formalize the experience and best practices, we have found a completely different approach to knowledge representation, which is the rule based representation. Basically, we translate the pieces of advice and best practices into Event Condition Action (ECA [Huebscher and McCann (2008)]) rules. Such rules are often associated with business intelligence practices, and integrating different rules at different timelines (via the autonomic aspect) proved to be a good choice for our proposition. ECA rules have certain drawbacks, such as it is hard to prove the coherence and the no contradiction. But for the rules in our system, this aspect is not currently an issue.

3 Driving the data warehouse – Autonomic Computing

Once the principal knowledge types are well separated and formalized, they have to be 'put to life'. We refer of course at the second aspect of the improvement system: rendering it autonomic. Autonomic systems have been present within our everyday lives. A very intuitive example of an autonomic system that manages itself is the human body. Reflexes like breathing, digestion, heart pulsation etc. are part of the autonomy the human body provides (we don't control these we just know they are continually present and moreover they function). Starting from this idea, the first approaches were especially towards self-healing systems, the survey of [Ghosh et al.(2007)] summing up this evolution. And, as expected the concept developed, and in 2001, IBM proposed a formalization of the self-x factor by introducing the notion of Autonomic Computing (AC) [IBM (2001)]. Most of the IT organizations spend a lot of time reacting to problems that occur at the IT infrastructure component level. This prevents them from focusing on monitoring their systems and from being able to predict and prevent problems before end users are impacted [IBM (2005)]. Autonomic computing is the ability for an IT infrastructure to adapt and change in accordance with business policies and objectives. Quite simply, it is about freeing IT professionals to focus on higher–value tasks by making technology work smarter, with business rules guiding systems to be self-configuring, self-healing, self-optimizing and self-protecting [IBM (2001)].

From this to applying autonomic computing to enable improvement in IT infrastructures was just a small step. The subject proved to be of great interest to enterprises. Works have been done in this area and put into practice for improving database performance by IBM [Markl et al.(2003)], [Lightstone et al.(2002)] and Microsoft [Mateen et al.(2008)]. IBM specifications link autonomic computing with the notion of autonomic manager as the entity that coordinates the activity of the autonomic process. An autonomic manager (ACM) is an implementation that automates the self-management function and externalizes this function according to the behavior defined by management interfaces. The autonomic manager is a component that implements an intelligent control loop. For a system component to be self-managing, it must have an automated method to collect the details it needs from the system (Monitor); to analyze those details to determine if something needs to change (Analyze); to create a plan, or sequence of actions, that specifies the necessary changes (Plan); and to perform those actions (Execute) [IBM (2001)]. Similar alternatives to autonomic computing were made in real BI [Nguyen et al.(2005)] but the idea is the same: to be able to analyze and improve (in our case) a given system through a closed loop that differentiates a series of states.

We propose the usage of autonomic managers to enable data warehouse self-improvement. Figure 2 shows the transformation of the architecture from Figure 1, with the implementation of autonomic managers on each of the entities (or component of the architecture) involved.

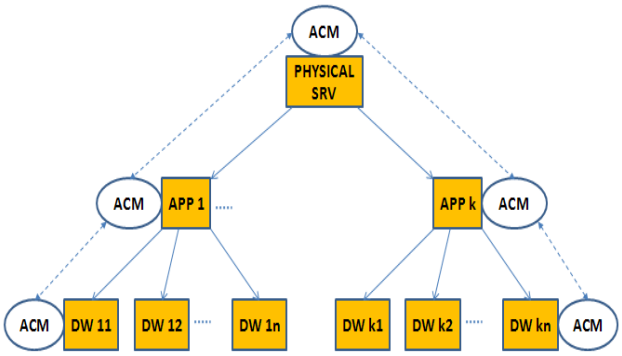


Figure 2 - Autonomic Computing Managers on each of the architectural levels

We notice that each of the entities has its own individual loop. The autonomic managers communicate only with the ones from the superior levels, and not between the same level. This way, each entity has two responsibilities: one strictly related to its individual self management and the other related to the management of its descendants. The idea is that the two can function independently of each other. For instance, consider an Application that has 2GB of RAM allocated to its data warehouses. So each data warehouse uses the allocated RAM and self-improves itself with what it has. Now suppose that at a certain point the Application receives another 1GB of RAM. If the new information is not integrated then the data warehouses continue to function with the already allocated 2GB. Once the application runs the management of its descendants, a reallocation of the memory is done also for the data warehouses. In order to simulate the two behaviors, we have elaborated two heuristics.

3.1 Data warehouse Self-improvement heuristic

This concerns only the individual loop at a data warehouse level. Its role is to describe how cache allocations vary with the query response times. The idea is the following: starting from a given maximal cache configuration we try to decrease the values of the caches and study the impact this decrease has on the data warehouse query response times. The algorithm stops when the difference between the current and the last average query response time is greater than a specified threshold. This is done independently for each data warehouse. So, we define two parameters for this heuristics:

Step - represents the amount with which each cache value is decreased. The following formula shows how a cache value modifies with step:

$$CV_1 = CV_0 - (CV_{max} - CV_0) * step$$

where CV_0 represents the old cache value, CV_1 the new calculated value and CV_{max} the maximum possible value . A frequent value of step we used in our experiments was 10% based on the recommendation of our human experts.

Delta – represents the threshold accepted for the difference between the current and the last average response time. It can be seen as the accepted impact that a cache modification has. If $(RT_1-RT_0)/RT_0 < delta$ then we accept the new cache proposition (where *RT = average response time* for the respective data warehouse). A frequent value we used for delta was 5%, based on our clients' average performance acceptance specification (i.e. for a value of x, an fluctuation in performance with 5% is accepted).

Table 1 illustrates the self-improvement heuristics with a timeline, based on the autonomic manager loop phases. At t_0 we have the initial configuration. At t_1 we have made the first cache adjustment, and validated it. At t_2, the second cache modification has an impact too great on the response time so we leave the cache value as it is.

Table 1 - Individual Data Warehouse Self-Improvement Heuristics on the autonomic manager phases

Time	AML Phase	Action
0	Monitor	step = 0.1, delta = 0.05, CV_{max}=1GB CV_0=500MB, RT_0=5s
	Analyze	N(ot)/A(vailable)
	Plan	CV_1=450MB
	Execute	Change script for DW with CV_1
1	Monitor	CV_1=450MB, RT_1=5.2s
	Analyze	$(RT_1-RT_0)/RT_0$=0.04 < delta
	Plan	CV_2=395MB
	Execute	Change script for DW with CV_2
2	Monitor	CV_2=395MB, RT_2=6s
	Analyze	$(RT_2-RT_1)/RT_1$=0.15 > delta
	Plan	CV_3=395MB
	Execute	No change for DW

3.2 Group of data warehouses cache reallocation heuristic

The first heuristics was individual data warehouse based. Each of the data warehouses was independent and each was in a state of self-improvement in time. But, taking it into consideration alone makes no sense as the performances on individual data warehouses are expected to decrease as the caches decrease. To explain how it results in an actual improvement at group level, we introduce the group of data warehouses heuristic. Its purpose is to reallocate periodically the memory that the individual data warehouse heuristics saved from the self-improvement process. And it is here where the 'catch' is: by a small sacrifice (delta) of some data warehouses, we can gain an important performance on others.

The core of the heuristic is to differentiate the non performing from the performing data warehouses in a group. The idea is the following: a data warehouse is considered performing if its average response time is below the average value of the response time for the whole group. Otherwise, it is considered as non-performing. This performance indicator can be equally made more complex by taking into account the applications priority or importance. This way scaled mixed performance indicators can be obtained and used. The specification of priorities and importance is usually part of Service License Agreements and is one of the future directions in our work.

So in this case, we take the memory from the performing data warehouse and give it to the non-performing. Relating with the architecture in Figure 1, the Application level is responsible for the implementation of this heuristic. The Application decides how to redistribute the memory between the data warehouses it concerns.

Table 2 shows this heuristics. The example is based on a group of two data warehouses that are part of the same application and share the same amount of memory for their caches.

Table 2 - Group of Data Warehouse Improvement Heuristic – Cache Evolution Exemple

Step	DW	Cache Value	Memory to allocate	Free Memory	RT
0	DW1	130 MB	140 MB	10 MB	5s
	DW2	80 MB	90 MB	10 MB	7s
1	DW1	130 MB	130 MB	0 MB	5s
	DW2	100 MB	100 MB	0 MB	6s
2	DW1	120 MB	120 MB	0 MB	5.3s
	DW2	110 MB	110 MB	0 MB	5.5s

In start at s_0 we have a given cache allocation along with the available memory for each data warehouse. At s_1 the heuristic is run the first time. It takes all the available memory from the performing data warehouse (DW1) and redistributes it to the non performing (DW2). So DW2 gains all the free memory (20MB) from s_0. As the differences in response times are still important, it goes further at s_2. Here, it takes some memory from DW1 by force, leading to a decrease in performance for DW2. But as seen, we gain an important amount of performance for DW2, and now the response times for the two data warehouses are close.

It is important to note that this heuristic is independent from the previous one, and in addition the two heuristics are mutually exclusive. This means that in the moments when this heuristic is considered, the other does nothing. This is why between the two tables we differentiate between "Time" and "Step". An example of usage is to run the individual self-improvement heuristic once each day (from Tuesday to Friday), and the group reallocation heuristic once at the beginning of each weak (Monday).

4 Combining the elements

Having the two main aspects, knowledge formalization and autonomic capabilities, the final and innovative stage in our approach is to combine them. In order to do this we base on the preliminary works presented in [[Nicolicin-Georgescu et al.(2009)]]. The solution proposed the application of ontologies and ontology based rules (describing business rules) with autonomic computing for improving

average query response times in data warehouse. The concept is the same, but in this previous work we only described how can simple businesses rules can be used to improve data warehouse performance. There is no indication to how heuristics are used within the autonomic manager loop.

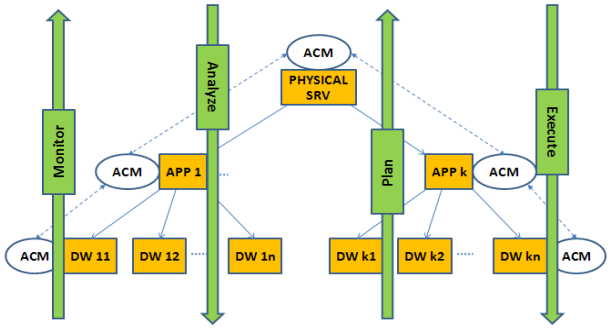
4.1 System implementation

In previous works, we were proposing a division of the knowledge in the system into static knowledge and dynamic knowledge. Based on this organization we implement the new presented elements. The means of knowledge formalization do not change, static knowledge being implemented with the help of ontologies and OWL[3] whereas the dynamic aspect is expressed with ontology based rules via the Jena Rules[4] formalization. The ontology contains over 150 concepts and 250 axioms, whereas a number of 30 rules are based on it. From what we have presented, we focus on the dynamic aspect, as it includes the two heuristics projected on the phases of the autonomic computing.

The first step in order to understand how rules are organized is to understand how the autonomic managers on the different hierarchical levels communicate. As seen the group heuristic reallocates memory and excludes the individual heuristic. In order for the autonomic managers to communicate, we propose a hierarchy of the autonomic phases, corresponding to the architectural structure. Figure 3 show how the four phases of autonomic manager are projected on the architectural levels.

We notice how the monitor phases ascends, starting from the lowest level (data warehouse). This means that first the data regarding the data warehouses are gathered, then the application, and then the physical server. Then the analysis is made top down from the physical server to the data warehouse level. Retaking the memory allocation example, first the server allocates memory between its applications, then each application allocates in turn to its data warehouses etc. Then the planning stage ascends again, the changes are planned from the analysis level starting with the data warehouses and finishing with the physical server. Last, the execution phase makes changes top down similar to the analyze phase. A change in the RAM memory is first done to the physical server, then the applications receive the new memory and then the data warehouses change their memory (now possible because the memory has already been changed at application level).

Figure 3 - Autonomic Manager phases projection on the architectural levels



[3]http://www.w3.org/2007/OWL/wiki/OWL_Working_Group

[4] http://jena.sourceforge.net/inference/#rules

We exemplify below how the system is implemented on each of the four phases.

Monitor

For the monitor phase, in order to obtain the cache values and average response times, we use SQL data bases that are filled with the help of vbscripts via the api provided by Hyperion Essbase. Then, to transform and load this knowledge in the ontology, we pass via a java program using the Jena API and a set of correspondences that links the data from the SQLdbs to ontology concepts. Table 3 shows how some parts of how a data warehouse is represented in the ontology:

Table 3 - Data Warehouse ontology representation

Subject	Predicate	Object
?dw	rdf:type	DataWarehouse
?app	rdf:type	PhysicalApplication
?dw	isChildOf	?app
?dw	hasAvgResponseTime	?avgt
?dw	hasPrevAvgResponseTime	?prevt

We can see two classes, the DataWarehouse and the PhysicalApplication. Each of these classes consist from multiple instances as OWL individuals. The ?dw is one such individual that is linked to an ?app individual by the OWL object property isChildOf . This property establishes the hierarchical relations between individuals from the different hierarchical levels. Then, there are two OWL data type properties that are linked to the ?dw and express the current and previous average response time for ?dw. The values for these properties are filled from the SQL dbs that contains to the data warehouse monitor information.

Analyze

Once this phase of monitoring and pre-processing of information is done, the system passes to analyze. We present below two rules that formalize a cache decrease.

Rule	Description
(?dw cp:hasPrevAvgResponseTime ?prevt) (?dw cp:hasAvgResponseTime ?avgt) (?dw cp:hasAlgorithm ?alg) (?alg cp:hasDelta ?delta) quotient(?t, ?avgt, ?prevt) le(?t, ?delta) ->(?dw cp:hasState cp:DecreaseCache)	Validate a cache decrease via the individual heuristic
(?dw cp:hasState cp:DecreaseCache) (?dw cp:hasIndexCacheMin ?ic_min) (?dw cp:hasIndexCache ?ic) (?dw cp:hasAlgorithm ?alg) (?alg cp:hasStep ?step) product(?p, ?ic, ?step) difference(?ic_new, ?ic, ?p) ge(?ic_new, ?ic_min) ->(?dw cp:hasIndexCache ?ic_new)	If the decrease of cache is requested, test if the new value is not under the minimal value. If not enable the new change.

The first rule test to whether the cache values for a single data warehouse can be decreased, accordingly with the individual heuristic. We have again the ?dw individual, an instance of the DataWarehouse class, with the two data type properties from Table 4. In addition we have a new object property that related the ?dw with an the individual heuristic algorithm. The rule compares the rapport between the two average times (current and previous) with the delta of the algorithm. If the rapport is lower than delta, the ?dw becomes into a new state, in which it is allowed to decrease its cache. Otherwise, nothing changes.

The second rule makes use of the results of the first rule. If the DecreaseCache state has been generated by the first rule, then it tries to see whether or not the operation is possible. Two new data type properties are introduced for the ?dw: hasIndexCacheMin and hasIndexCache, which represent the values of the minimum threshold and the current value of the cubes index cache. These values are equally filled from the monitor phase. The rule retakes the individual heuristic algorithms' parameters (step this time) and tests if by modifying the index cache with its formula, the new index cache value is greater than the minimum one. If so, it changes directly the current index cache value to the newly computed one.

Plan and Execute

The plan and execute phases are linked to each other. As the new cache values are calculated, there is a preparation of VBscripts that will be run via the program. These scripts will change the values of the caches in the Essbase cubes, according to the new values proposed by the analyze phase. At the end of the execution phase, practically the inverse monitor operation of data processing is made. The cache values are passed from the ontology to the SQLdbs and then to the modification scripts.

4.2 Experimentation and Results

For our experiments we considered the following scenario: on an existing server, we created an Essbase application with two cubes. The cubes contain in average 11 principal axes and 27 level 2 axes and the data file has an average size of 300MB. With this configuration, we carried several tests, simulating a period of 14 days (time stamps period). Each time-stamp, a series of random queries (from a given set) was executed so that activity on the application was simulated. The individual data warehouse self improvement heuristic is running each day, whereas the group heuristic is running once each 4 days. Figure 4 shows the evolution of the response times for the two data warehouses with the evolution of their total cache allocation:

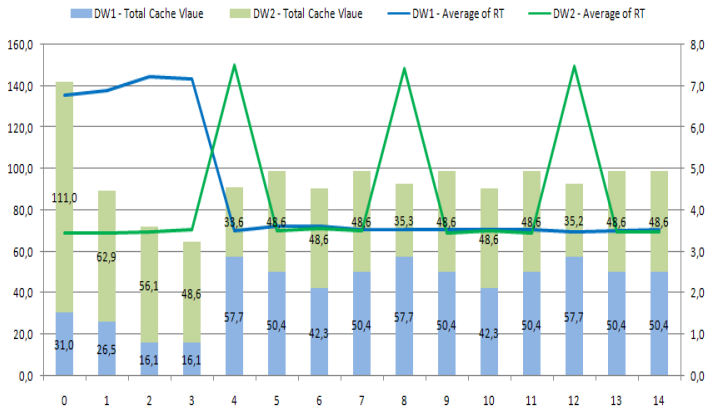


Figure 4 - Average Response Time evolution with cache allocations

Again, the objective is to obtain better average response times with lower cache values. First what we notice is that at the end of 5 days we already have a good ratio response time/cache allocation. The data warehouses improve themselves fast, and then once reaching a good point, they oscillate around this point. This oscillation is shown by the peaks on DW2 that tries each time to improve more its performances, but it can't due to the heuristic constraints in terms of delta. Their impact on the system is felt in terms of the performance drops the days where the peaks are noticed. By limiting the number of peaks (i.e. after one or two peaks the system should no longer try to optimize under the same circumstances) we avoid the risk of such drops. But, we also have to take into consideration that by limiting the number of peaks (by forcing the algorithm to stop for instance at a certain point) we risk to miss some needs of improvement due to reconfiguration aspects. The ideal would be to leave the heuristic running as usual and not to force the algorithm to stop, but not to accept the cache decreases once a certain level of performance is reached. One of the future directions and improvements is the introduction of attenuation mechanisms in the loop.

So, in numbers, at the end of the sixth day: DW1 looses very little in performances (~2%), DW2 gains substantial performances (~80%), and the total cache used by the application is decreased by ~60%. So the sacrifice of DW1 was worth from the perspective of the entire system. These results prove how an efficient way of improving the data warehouse group performances can be achieved in an autonomic manner, without the intervention of a human expert.

5 Conclusions

This article presented a way of using ontologies and autonomic computing for improving query response times in data warehouses groups by modifying cache memory allocations. It has presented this applied to the problematic of shared resource allocation in groups of data warehouses. Also, the article presented a proposition of replacing some of the human expert's work by introducing autonomic and human independent ways of managing data warehouses.

We have proposed a division and formalization of the knowledge used for configuring groups of data warehouses by using ontology and ontology based rules. Also, we have proposed an organization of this process based on the autonomic computing considerations. It is not the first attempt to combine the two [Stojanovic et al.(2004)], but the novelty is from using such techniques in the domain of decision support systems and especially in the groups of data warehouse improvement.

Our future directions are to expand the data warehouses described above so that our prototype can prove its efficiency on a larger spectrum of rules and indicators. Our purpose is to integrate the prototype presented here with more than one aspect (data warehouse cache allocations based on response times) of decision support systems. We also intend to approach the notions of Service License Agreement (SLA) and Quality of Service (QoS), by introducing the QoS as a performance indicator in the system. SLA considerations such as application priority and importance depending on utilization periods, are two aspects that are little approached and equally very important in a decision support system. Also in terms of autonomic loop control, we take into consideration the usage of mechanisms for avoiding peaks and unnecessary loop passages.

As the domain is relatively new we try to bring as much support as possible for future development in the direction of autonomic knowledge based decision support systems. We follow the changes with the new technologies and hope that our work will be useful in this expanding environment.

References

[Barret *et al.*(2004)] Rob Barret, Paul P. Maglio, Eser Kandogan, and John Bailey. Usable autonomic computing systems: the administrator's perspective. In *ICAC 2004*, 2004.

[Druzdel and Flynn (1999)]M.J. Druzdel and R.R. Flynn. *Decision Support Systems*. Encyclopedia of library and information science, 1999.

[Frolick and Lindsey (2003)] Mark N. Frolick and Keith Lindsey. Critical factors for data warehouse failure. *Business Intelligence Journal*, Vol. 8, No. 3, 2003.

[Ghosh *et al.*(2007)] Debanjan Ghosh, Raj Sharman, H. Raghav Rao, and Shambhu Upadhyaya. Self-healing systems — survey and synthesis. *Decision Support Systems 42*, Vol 42:p. 2164–2185, 2007.

[Gruber (1992)] T. Gruber. *What is an ontology?* Academic Press Pub., 1992.

[Huebscher and McCann (2008)] M.C. Huebscher and J.A. McCann. A survey on autonomic computing – degrees, models and applications. *ACM Computing Surveys*, Vol. 40, No. 3, 2008.

[IBM (2001)] Corporation IBM. *An architectural blueprint for autonomic computing*. IBMCorporation, 2001.

[IBM (2005)] Corporation IBM. Autonomic computing. powering your business for success. *International Journal of Computer Science and Network Security*, Vol.7 No.10:p. 2–4, 2005.

[Inmon (2005)] W.H. Inmon. *Building the data warehouse, fourth edition*. Wiley Publishing, 2005.

[Lightstone *et al.*(2002)] S.S. Lightstone, G. Lohman, and D. Zilio. Toward autonomic computing with db2 universal database. *ACM SIGMOD Record*, Vol. 31, Issue 3, 2002.

[Malik *et al.*(2008)]T. Malik, X. Wang, R. Burn, D. Dash, and A. Ailamaki. Automated physical design in database caching. In *ICDE Workshop*, 2008.

[Manoel *et al.*(2005)] E. Manoel, M.J. Nielsen, A. Salahshour, S. Sampath, and S. Sudarshanan. Problem determination using self-managing autonomic technology. *IBM RedBook*, pages p. 5 – 9, 2005.

[Markl *et al.*(2003)]V. Markl, G. M. Lohman, and V. Raman. Leo : An autonomic optimizer for db2. *IBM Systems Journal*, Vol. 42, No. 1, 2003.

[Mateen *et al.*(2008)] A. Mateen, B. Raza, and T. Hussain. Autonomic computing in sql server. In *7th IEEE/ACIS International Conference on Computer and Information Science*, 2008.

[Nguyen *et al.*(2005)] T. M. Nguyen, J. Schiefer, and A. Min Tjoa. Sense & response service architecture (saresa). In *DOLAP'05*, 2005.

[Nicolicin-Georgescu *et al.*(2009)] Vlad Nicolicin-Georgescu, Vincent Benatier, Remi Lehn, and Henri Briand. An ontology-based autonomic system for improving data warehouse performances. In *Knowledge-Based and Intelligent Information and Engineering Systems, 13th International Conference, KES2009*, 2009.

[Saharia and Babad (2000)]A. N. Saharia and Y.M. Babad. Enhancing data warehouse performance through query caching. *The DATA BASE Advances in Informatics Systems*, Vol 31, No.3, 2000.

[Stojanovic *et al.*(2004)] L. Stojanovic, J. Schneider, A. Maedche, S. Libischer, R. Studer, Th. Lumpp, A. Abecker, G. Breiter, and J. Dinger. The role of ontologies in autonomic computing systems. *IBM Systems Journal*, Vol. 43, No. 3:p. 598–616, 2004.

Barret *et al.*(2004)] Rob Barret, Paul P. Maglio, Eser Kandogan, and John Bailey. Usable autonomic computing systems: the administrator's perspective. In *ICAC 2004*, 2004.

[Druzdel and Flynn (1999)]M.J. Druzdel and R.R. Flynn. *Decision Support Systems*. Encyclopedia of library and information science, 1999.

[Frolick and Lindsey (2003)] Mark N. Frolick and Keith Lindsey. Critical factors for data warehouse failure. *Business Intelligence Journal*, Vol. 8, No. 3, 2003.

[Ghosh *et al.*(2007)] Debanjan Ghosh, Raj Sharman, H. Raghav Rao, and Shambhu Upadhyaya. Self-healing systems — survey and synthesis. *Decision Support Systems 42*, Vol 42:p. 2164–2185, 2007.

[Gruber (1992)] T. Gruber. *What is an ontology?* Academic Press Pub., 1992.

[Huebscher and McCann (2008)] M.C. Huebscher and J.A. McCann. A survey on autonomic computing – degrees, models and applications. *ACM Computing Surveys*, Vol. 40, No. 3, 2008.

[IBM (2001)] Corporation IBM. *An architectural blueprint for autonomic computing*. IBMCorporation, 2001.

[IBM (2005)] Corporation IBM. Autonomic computing. powering your business for success. *International Journal of Computer Science and Network Security*, Vol.7 No.10:p. 2–4, 2005.

[Inmon (2005)] W.H. Inmon. *Building the data warehouse, fourth edition*. Wiley Publishing, 2005.

[Lightstone *et al.*(2002)] S.S. Lightstone, G. Lohman, and D. Zilio. Toward autonomic computing with db2 universal database. *ACM SIGMOD Record*, Vol. 31, Issue 3, 2002.

[Malik *et al.*(2008)]T. Malik, X. Wang, R. Burn, D. Dash, and A. Ailamaki. Automated physical design in database caching. In *ICDE Workshop*, 2008.

[Manoel *et al.*(2005)] E. Manoel, M.J. Nielsen, A. Salahshour, S. Sampath, and S. Sudarshanan. Problem determination using self-managing autonomic technology. *IBM RedBook*, pages p. 5 – 9, 2005.

[Markl *et al.*(2003)]V. Markl, G. M. Lohman, and V. Raman. Leo : An autonomic optimizer for db2. *IBM Systems Journal*, Vol. 42, No. 1, 2003.

[Mateen *et al.*(2008)] A. Mateen, B. Raza, and T. Hussain. Autonomic computing in sql server. In *7th IEEE/ACIS International Conference on Computer and Information Science*, 2008.

[Nguyen *et al.*(2005)] T. M. Nguyen, J. Schiefer, and A. Min Tjoa. Sense & response service architecture (saresa). In *DOLAP'05*, 2005.

[Nicolicin-Georgescu *et al.*(2009)] Vlad Nicolicin-Georgescu, Vincent Benatier, Remi Lehn, and Henri Briand. An ontology-based autonomic system for improving data warehouse performances. In *Knowledge-Based and Intelligent Information and Engineering Systems, 13th International Conference, KES2009*, 2009.

[Saharia and Babad (2000)]A. N. Saharia and Y.M. Babad. Enhancing data warehouse performance through query caching. *The DATA BASE Advances in Informatics Systems*, Vol 31, No.3, 2000.

[Stojanovic *et al.*(2004)] L. Stojanovic, J. Schneider, A. Maedche, S. Libischer, R. Studer, Th. Lumpp, A. Abecker, G. Breiter, and J. Dinger. The role of ontologies in autonomic computing systems. *IBM Systems Journal*, Vol. 43, No. 3:p. 598–616, 2004.